A Fast-but-Gentle Introduction to Artificial Intelligence Acceleration

Dr. Francesco Conti, DEI & ARCES
f.conti@unibo.it
Deep NN Timeline

- **1940s**: Neural networks were proposed
- **1960s**: Deep neural networks were proposed
- **1989**: Neural network for recognizing digits (LeNet)
- **1990s**: Hardware for shallow neural nets
  - Example: Intel ETANN (1992)  NVIDIA GPUs with CUDA available
- **2011**: Breakthrough DNN-based speech recognition
  - Microsoft real-time speech translation
- **2012**: DNNs for vision supplanting traditional ML
  - AlexNet for image classification
- **2014+**: Rise of DNN accelerator research
  - Examples: Neuflow, DianNao, etc.
Deep ConvNets (depth inflation)

**VGG**
[Simonyan 2013]

**GoogLeNet**
[Szegedy 2014]

**ResNet**
[He et al. 2015]

**DenseNet**
[Huang et al. 2017]
Searching for "AI" on Google Image Search

[CC images, various sources]
Searching for "AI" on Google Image Search

Ok, not much information here!
Searching for "Deep Neural Network" on Google Image Search

Convolutional Network with Feature Layers

Input Layer
Hidden Layer
Output Layer
Searching for "Deep Neural Network" on Google Image Search

Convolutional Network with Feature Layers

Much better! But still not crystal clear.

Input Layer
Hidden Layer
Output Layer

[CC images, various sources]
3.1 Encoder and Decoder Stacks

Encoder: The encoder is composed of a stack of \( N = 6 \) identical layers. Each layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, position-wise fully connected feed-forward network.

We employ a residual connection \([11]\) around each of the two sub-layers, followed by layer normalization \([1] \).

That is, the output of each sub-layer is \( \text{LayerNorm}(x + \text{Sublayer}(x)) \), where \( \text{Sublayer}(x) \) is the function implemented by the sub-layer itself.

To facilitate these residual connections, all sub-layers in the model, as well as the embedding layers, produce outputs of dimension \( d_{\text{model}} = 512 \).

Decoder: The decoder is also composed of a stack of \( N = 6 \) identical layers. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack.

Similar to the encoder, we employ residual connections around each of the sub-layers, followed by layer normalization. We also modify the self-attention sub-layer in the decoder stack to pre-vent positions from attending to subsequent positions.

This masking, combined with the fact that the output embeddings are offset by one position, ensures that the predictions for position \( i \) can depend only on the known outputs at positions less than \( i \).

3.2 Attention

An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.
Deep Learning from the Eye of an Accelerator Architect

1 element = 1 number
Deep Learning from the Eye of an Accelerator Architect

1 cube = many elements in 3D

1 element = 1 number
Deep Learning from the Eye of an Accelerator Architect

multiplications & additions
Deep Learning from the Eye of an Accelerator Architect

1 cube - many elements in 3D

1 element = 1 number

many-to-one dependency (in general)

LINEAR TRANSFORMATION

modified shape

multiplications & additions
Deep Learning from the Eye of an Accelerator Architect

1 cube - many elements in 3D

1 element = 1 number

many-to-one dependency (in general)

LINEAR TRANSFORMATION

modified shape

multiplications & additions

comparisons, exponentials, ...

NON-LINEARITY
Deep Learning from the Eye of an Accelerator Architect

1 cube - many elements in 3D

1 element = 1 number

multiplications & additions

many-to-one dependency (in general)

 LINEAR TRANSFORMATION

modified shape

comparisons, exponentials, ...

one-to-one dependency

NON-LINEARITY
Deep Learning from the Eye of an Accelerator Architect

Infinite compositions, but “basic ingredients” are all here! Simple ideas are applicable to wide variety of models: DNNs, RNNs, Transformers...

- Linear transformation
- Many-to-one dependency (in general)
- One-to-one dependency
- Addition, concatenation...
- Multiplications & additions
- Comparisons, exponentials, ...
Deep Learning from the Eye of an Accelerator Architect

1. how to **represent** data
2. how to **compute** data transformations
3. where to **store** data and how to **move** them around
1. how to represent data

<table>
<thead>
<tr>
<th>Method</th>
<th>Data Range</th>
<th>Parameter Range</th>
<th>ImageNet Top1 Acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full-Precision</td>
<td>Real numbers (FP32)</td>
<td>Real numbers (FP32)</td>
<td>69.6</td>
</tr>
<tr>
<td>Linear Quantization</td>
<td>Integers 0 to 255</td>
<td>Integers -128 to +127</td>
<td>69.6</td>
</tr>
<tr>
<td>PACT</td>
<td>Integers 0 to 15</td>
<td>Integers -8 to +7</td>
<td>69.2</td>
</tr>
<tr>
<td>PACT</td>
<td>Integers 0 to 7</td>
<td>Integers -4 to +3</td>
<td>68.1</td>
</tr>
<tr>
<td>PACT</td>
<td>Integers 0 to 3</td>
<td>Integers -2 to +1</td>
<td>64.4</td>
</tr>
<tr>
<td>PACT</td>
<td>Integers 0 to 3</td>
<td>-1 or +1</td>
<td>62.9</td>
</tr>
<tr>
<td>XNOR-Net</td>
<td>-1 or +1</td>
<td>-1 or +1</td>
<td>51.2</td>
</tr>
</tbody>
</table>
Deep Learning from the Eye of a Accelerator Architect

1. how to **represent** data

<table>
<thead>
<tr>
<th>Method</th>
<th>Data Range</th>
<th>Parameter Range</th>
<th>ImageNet Top1 Acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full-Precision</td>
<td>Real numbers (FP32)</td>
<td>Real numbers (FP32)</td>
<td>69.6</td>
</tr>
<tr>
<td>Linear Quantization</td>
<td>Integers 0 to 255</td>
<td>Integers -128 to +127</td>
<td>69.6</td>
</tr>
<tr>
<td>PACT</td>
<td>Integers 0 to 15</td>
<td>Integers -8 to +7</td>
<td>69.2</td>
</tr>
<tr>
<td>PACT</td>
<td>Integers 0 to 7</td>
<td>Integers -4 to +3</td>
<td>68.1</td>
</tr>
<tr>
<td>PACT</td>
<td>Integers 0 to 3</td>
<td>Integers -2 to +1</td>
<td>64.4</td>
</tr>
<tr>
<td>PACT</td>
<td>Integers 0 to 3</td>
<td>-1 or +1</td>
<td>62.9</td>
</tr>
<tr>
<td>XNOR-Net</td>
<td>-1 or +1</td>
<td>-1 or +1</td>
<td>51.2</td>
</tr>
</tbody>
</table>

2. how to **compute** data transformations

- dominated by simple arithmetic operations
- many-to-one $\rightarrow$ many possible compute orderings
- independent operations $\rightarrow$ can be done in parallel / hierarchically
- hardware complexity / speed /energy related to representation
Deep Learning from the Eye of a Accelerator Architect

1. how to **represent** data

<table>
<thead>
<tr>
<th>Method</th>
<th>Data Range</th>
<th>Parameter Range</th>
<th>ImageNet Top1 Acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full-Precision</td>
<td>Real numbers (FP32)</td>
<td>Real numbers (FP32)</td>
<td>69.6</td>
</tr>
<tr>
<td>Linear Quantization</td>
<td>Integers 0 to 255</td>
<td>Integers -128 to +127</td>
<td>69.6</td>
</tr>
<tr>
<td>PACT</td>
<td>Integers 0 to 15</td>
<td>Integers -8 to +7</td>
<td>69.2</td>
</tr>
<tr>
<td>PACT</td>
<td>Integers 0 to 7</td>
<td>Integers -4 to +3</td>
<td>68.1</td>
</tr>
<tr>
<td>PACT</td>
<td>Integers 0 to 3</td>
<td>Integers -2 to +1</td>
<td>64.4</td>
</tr>
<tr>
<td>PACT</td>
<td>Integers 0 to 3</td>
<td>-1 or +1</td>
<td>62.9</td>
</tr>
<tr>
<td>XNOR-Net</td>
<td>-1 or +1</td>
<td>-1 or +1</td>
<td>51.2</td>
</tr>
</tbody>
</table>

2. how to **compute** data transformations

i. dominated by simple arithmetic operations

ii. many-to-one $\rightarrow$ many possible compute orderings

iii. independent operations $\rightarrow$ can be done in parallel / hierarchically

iv. hardware complexity / speed / energy related to representation

3. where to **store** data and how to **move** them around

The main source of headaches for DL Accelerator Architects!
A Minimal Accelerator

**COMPUTE:**

- Memory Read
- Multiply & Accumulate
- Memory Write

- weight
- input tensor
- partial sum
- updated partial sum

**MEMORY:**

- Off-Chip $>10^{-9}$J

**Non-Von Neumann... see the other talk!**
Memory Access is the Bottleneck

Memory Read  MAC*  Memory Write

DRAM  ALU  DRAM

* multiply-and-accumulate

Off-Chip >10⁻⁹J

Worst Case: all memory R/W are DRAM accesses

AlexNet [NIPS 2012] has 724M MACs → 2896M DRAM accesses required

DRAM access 100-1000x less energy-efficient than on-chip access!
Memory Access is the Bottleneck

Memory Read | MAC* | Memory Write

DRAM | ALU | DRAM

* multiply-and-accumulate

Off-Chip >10^-9J

Example: Off-Chip >10^-9J
Memory Access is the Bottleneck

Extra levels of local memory hierarchy

Off-Chip >10^{-9}J

On-Chip ~10^{-11}J
Memory Access is the Bottleneck

Reduce memory cost by error resilience
Memory Access is the Bottleneck

Memory Read | MAC | Memory Write

DRAM | Mem | ALU | Mem | DRAM

Reduce memory cost by **error resilience**

Reduce transfer cost by **data tiling**
Memory Access is the Bottleneck

Reduce memory cost by **error resilience**

Reduce transfer cost by **data tiling**

Reduced precision

[Graph showing Bit Error Rate vs. Vdd]

[Diagram showing Memory Read, MAC, Memory Write]
The Recipe for a Deep Learning Accelerator

1. Many Multiply-Accumulate (MAC) units to exploit parallelism

2. Flexible or Customized on-chip memory organization to keep as much data as possible on-chip, maximise its reuse...

3. Keep track of all external memory transfer overheads!
The Recipe for a Deep Learning Accelerator

1. Many **Multiply-Accumulate (MAC)** units to exploit parallelism

2. Flexible or Customized **on-chip memory organization** to keep as much data as possible on-chip, maximise its reuse…

3. Keep track of all **external memory** transfer overheads!

*Custom architectures*

**Google Cloud TPU**  **Intel/Movidius Myriad X**

**Tesla FSF Chip**

- LPDDR4 128b @4266Gbps → 68GBps
- 12ARM Cores
- Neural network Processor
The Recipe for a Deep Learning Accelerator

A12 (iPhone XS) – 7nm

How many processors? A lot!

CPU: 2/4 “big” cores + 4 “small” cores
GPU: 4/6 cores
Accelerators: Neural Processing Unit (NPU) + Image Signal Processor (ISP)
MCUs: Control + Always-On
The Queen of Deep Learning Accelerators: the GPU
The Queen of Deep Learning Accelerators: the GPU

NVIDIA GV100

5120 cores
80 multiprocessors
6MB L2 cache per chip
128KB L1 cache per multiprocessor

Tensor Cores
GPU Tensor Cores

TENSOR CORE
Mixed Precision Matrix Math
4x4 matrices

$$D = \begin{pmatrix}
A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\
A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\
A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\
A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3}
\end{pmatrix}$$

$$D = \begin{pmatrix}
B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\
B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\
B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\
B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3}
\end{pmatrix}$$

$$D = \begin{pmatrix}
C_{0,0} & C_{0,1} & C_{0,2} & C_{0,3} \\
C_{1,0} & C_{1,1} & C_{1,2} & C_{1,3} \\
C_{2,0} & C_{2,1} & C_{2,2} & C_{2,3} \\
C_{3,0} & C_{3,1} & C_{3,2} & C_{3,3}
\end{pmatrix}$$

$$D = AB + C$$
There’s Plenty of Room at the Bottom

• The relationship between data representation, network topology, and perf/energy/memory is not yet fully explored (particularly for tiny devices)!

• The Von Neumann model could be suboptimal: is it possible to sidestep memory in doing Multiply-Adds?

• Will future sophisticated AI algorithms show the same “good” properties of DNNS: regularity, parallelism, resilience?

[Lin et al., MCUNet: Tiny Deep Learning on IoT Devices]
Prof. Francesco Conti

DEI – Università di Bologna

f.conti@unibo.it